Chapter 4 Efficient Deterministic Voting in Distributed Systems

4.1 Introduction

In distributed storage systems, particularly for distributed file systems, voting can be used to keep replicated data consistent. Distributed voting is itself an important problem in the creation of fault-tolerant computing systems, e.g., it can be used to keep distributed data consistent and to provide mutual exclusion in distributed systems. In an N Modular Redundant (*NMR*) system, when the N computational modules execute identical tasks, they need to be synchronized periodically by voting on the current computation state (or result, and they will be used interchangeably hereafter), and then all modules set their current computation state to the majority one. If there is no majority result, then other computations are needed, e.g., all modules recompute from the previous result. This technique is also an essential tool for task-duplication-based checkpointing[38].

Many aspects of voting algorithms have been studied, e.g., data approximation, reconfigurable voting, dynamic modification of voting weights, and metadata-based dynamic voting[9][18][24]. In this chapter, we focus on the *communication complexity* of the voting problem. Several voting algorithms have been proposed to reduce the *communication complexity*[10][20]. These algorithms are nondeterministic because they perform voting on signatures (hash functions) of local computation results. Recently, a majority voting scheme based on error-control codes[21] was proposed: each module first encodes its local result into a codeword of a designed error-detecting code and sends part of the codeword. By using the error-detecting code, discrepancies of the local results can be detected with some probability, and then by retransmitting the full local results, a majority voting decision can be made. Though the scheme drastically reduces the *average case* communication complexity, it can still fail to detect some discrepancies of the local results and might reach a *false* voting result, i.e., the algorithm is still a probabilistic one. In addition, this scheme uses only the error-detecting capabilities of codes. As this chapter will show, in general, using only *error-detecting codes* (*EDC*) does not help to reduce the communication complexity of a deterministic voting algorithm. Though they have been used in many applications such as reliable distributed data replication[1], *error-correcting codes* (*ECC*) have not been applied to the voting problem.

For many applications[38], *deterministic* voting schemes are needed to provide more accurate voting results. In this chapter, we propose a novel *deterministic* voting scheme that uses error-correcting codes. The voting scheme generalizes all known simple deterministic voting algorithms. Our main contributions related to the voting scheme include: (i) using the correcting capability in addition to the detecting capability of codes (only the detection was used in known schemes) to drastically reduce the chances of retransmission of the whole local result of each node, thus reducing the communication complexity of the voting, (ii) a proof that our scheme reaches the same voting result as the naive voting algorithm in which every module broadcasts its local result to all the other modules, and (iii) a method of tuning the scheme for optimal average case communication complexity by choosing the parameters of the error-correcting code, thus making the voting scheme adaptive to various application environments with different error rates.

The chapter is organized as follows: in Section 4.2, we describe the majority voting problem in NMR systems. Our voting algorithm together with its correctness proof are described in Section 4.3. Section 4.4 analyzes both the worst case and the average case communication complexity of the algorithm. Section 4.5 presents experimental results of the performance of the proposed voting algorithm, as well as the performance of two other simple voting algorithms for comparison. Section 4.6 concludes the chapter.

4.2 The Problem Definition

In this section, we define the model of the NMR system and its communication complexity. Then we address the voting problem in terms of the communication complexity.

4.2.1 NMR System Model

An NMR system consists of N computational modules which are connected via a communication medium. For a given computational task, each module executes a same set of instructions with independent computational error probability p. The communication medium could be a bus, a shared memory, a point-to-point network or a broadcast network.

Here we consider the communication medium as a *reliable broadcast network*, i.e., each module can send its computational result to all the other modules with only one error-free communication operation. The system evolution is considered to be synchronous, i.e., the voting process is round-based.

4.2.2 Communication Complexity

The *communication complexity* of a task in an NMR system is defined as the *total* number of bits that are sent through the communication medium during the whole execution of the task. In a broadcast network, let m_{ij} be the number of the bits that the ith module sends at the jth round of the execution of a task, then the communication complexity of the task is $\sum_{i=1}^{N} \sum_{j=1}^{K} m_{ij}$, where N is the number of the modules in the system, and K is the number of rounds needed to complete the task.

4.2.3 The Voting Problem

Now consider the voting function in an NMR system. In order to get a final result for a given task in an NMR system, each module completes its own computation separately, then it must be synchronized with the other modules by voting on the result. Let X_i denote the local computational result of the ith module. The *majority function* is defined as follows:

$$Majority(X_1, \cdots, X_N) := \begin{cases} X & if \ \ |\{1 \leq i \leq N\} : X_i = X| \geq \frac{N+1}{2} \\ \phi & otherwise \end{cases}$$

where in general, N is an odd natural number, and ϕ is any predefined value different from all possible computing results.

Example 4.1 *Majority voting*

If $X_1 = 0000$, $X_2 = 0001$, $X_3 = 0100$, $X_4 = 0000$, $X_5 = 0000$, then

$$Majority(X_1, X_2, X_3, X_4, X_5) = 0000;$$

If X_5 changes to 0010, and the other $X_i's$ remain unchanged, then

$$Majority(X_1, X_2, X_3, X_4, X_5) = \phi.$$

□

The result of voting in an NMR system is that each module gets $Majority(X_1, \cdots, X_N)$ as its final result , where X_i $(i = 1, \cdots, N)$ is the local computation result of the ith module.

One obvious algorithm for the voting problem is that after each module computes the task, it broadcasts its own result to all the other modules. When a module receives all the other modules' results, it simply performs the majority voting locally to get the result. The algorithm can be described as follows:

Algorithm 4.1 *Send-All Voting*

For Module P_i, $i \in [1 : N]$:

Broadcast(X_i);

Wait Until Receive All X_j, $j \in [1 : N] \setminus i$;

$X := Majority(X_1, \cdots, X_N)$;

Return(X);

□

This algorithm is simple: each module only needs one communication (i.e., broadcast) operation, but apparently its communication complexity is too high. If the result for the task has m bits, then the communication complexity of the algorithm is Nm bits. In most cases, the probability of a module having a computational error is rather low, namely at most times all modules have the same result, thus each module only needs to broadcast part of its result. If all the results are identical, then each module agrees with that result. If not, then the modules can use *Algorithm 4.1*. In other words, we can improve the voting algorithm as follows:

Algorithm 4.2 *Simple Send-Part Voting*

For Module P_i, $i \in [1 : N]$:

Partition the local result X_i into N symbols: $X_i[1 : N]$;

Broadcast$(X_i[i])$;

Wait Until Receive All $X_j[j]$, $j \in [1 : N] \setminus i$;

$X := X_1[1] * \cdots * X_N[N]$;

$F_i := (X = X_i)$;

Broadcast(F_i);

If $Majority(F_1, \cdots, F_N) = TRUE(1)$

Return(X);

Else

Broadcast$(X_i[j])$, $j \in [1 : N] \setminus i$;

Wait Until Receive All X_j, $j \in [1 : N] \setminus i$;

Return$(Majority(X_1, \cdots, X_N))$;

□

In the above algorithm, * is a concatenation operation of strings, e.g., 000*100 = 000100; and = is an equality evaluation:

$$(X = Y) := \begin{cases} TRUE(1) & \text{if X equals Y} \\ FALSE(0) & \text{otherwise} \end{cases}$$

Some padding may be needed if the length of the local result is not an exact multiple of N. The following example demonstrates a rough comparison of the two algorithms.

Example 4.2 *Comparison of the voting algorithms*

Let $X_1 = X_2 = X_3 = X_4 = 00000$ and $X_5 = 10000$, *Algorithm 4.1* requires 1 round of communication, and transmits a total of 25 bits. On the other hand, with *Algorithm 4.2*, all P_i's $(i = 1, \cdots, 5)$ broadcast 0, and $X = 00000$. Thus $(F_1, \cdots, F_5) = 11110$, so $Majority(F_1, \cdots, F_5) = 1$, and X is the majority voting result. In this case, 2 rounds of communication are used, and 10 bits (5 bits for X and 5 bits for F) are transmitted.

If $X_5 = 00001$, and all the other X_i's remain same, then with *Algorithm 4.2*, $X = 00001$, which results in $(F_1, \cdots, F_5) = 00001$. Thus $Majority(F_1, \cdots, F_5) = 0$, and the *Else* part of the algorithm is executed. Finally, the majority voting result is obtained by voting on all the X_i's, i.e., $X = Majority(X_1, \cdots, X_5) = 00000$. Now 3 rounds of communication are needed, and in total, 30 bits (25 bits for the X_i's and 5 bits for F) are transmitted. □

From the above example, it can be observed that

1. *Algorithm 4.1* always requires only 1 round of communication, and *Algorithm 4.2* requires 2 or 3 rounds of communication;

2. The *Else* part of *Algorithm 4.2* is actually *Algorithm 4.1*;

3. The communication complexity of *Algorithm 4.1* is always Nm, but the communication complexity of *Algorithm 4.2* may be $m+N$ or $Nm+N$, depending on the X_i's;

4. In *Algorithm 4.2*, by broadcasting and voting on the *voting flags*, i.e., F_i's, the chance for getting a *false* voting result is eliminated.

If the *Else* part of *Algorithm 4.2*, i.e., *Algorithm 4.1*, is not executed too often, then the communication complexity can be reduced from Nm to $m+N$, and in most cases, $m \gg N$, thus $m+N \approx m$. So the key idea to reduce the communication complexity is to reduce the probability of executing *Algorithm 4.1*. In most computing environments, each module has a low computational error probability p, thus with high probability either *(1)* all modules get the same result or *(2)* only few of them get different results from the rest. In case *(1)*, *Algorithm 4.2* has low communication complexity, but in case *(2)*, *Algorithm 4.1* is actually used and the communication complexity is high (i.e., $Nm+N$). If we can detect and correct these few inconsistent results, then the *Else* part of the *Algorithm 4.2* does not need to be executed, and so the communication complexity can still be low. This detecting and correcting capability can be achieved by using error-correcting codes.

4.3 The Solution Based on Error-Correcting Codes

Error-correcting codes (*ECC*) can be used in the voting problem to reduce the communication complexity. The basic idea is that instead of broadcasting its own computation result X_i directly, the ith module, P_i, first encodes its result X_i into a codeword Y_i of some code, and then broadcasts one *symbol* of the codeword to all the other modules. After receiving all the other *symbols* of the codeword, it reassembles them into a vector. If all the modules have the same result, i.e., all the X_i's are equal, then the received vector is the codeword of the result, thus it can be decoded to retrieve the result. If a majority result exists, i.e., $Majority(X_1, \cdots, X_N) \neq \phi$, and there are t ($t \leq \lfloor \frac{N}{2} \rfloor$) modules whose results are different from the *majority result* X, then the *symbols* from all these modules can be regarded as error symbols with respect to the *majority result*. As long as the code is designed to correct up to t errors, these error symbols can be corrected to get the codeword corresponding to the majority result, thus *Algorithm 4.1* does not need to be executed. When the code length is less than Nm, the communication complexity is reduced compared to *Algorithm 4.1*. On the other hand, if only *error-detecting codes* are used, once errors are detected, *Algorithm*

4.1 still needs to be executed, and thus increases the whole communication complexity of the voting. Thus error-correcting codes are preferable to error-detecting codes for voting. By properly choosing the error-correcting codes, the communication complexity can *always* be lowered below that of *Algorithm 4.1*.

But it is possible that no *majority result* exists, i.e., $Majority(X_1, \cdots, X_N) = \phi$, yet the vector that each module gets can still be decoded. As in the example above, introducing voting flags can avoid this *false* result.

4.3.1 A Voting Algorithm with ECC

With a properly designed error-correcting code, which can detect up to d *error symbols* and correct up to t *error symbols* ($0 \leq t \leq d$), a complete voting algorithm using this code is as follows:

Algorithm 4.3 *ECC Voting*

For Module P_i, $i \in [1:N]$:

  $Y_i :=$ Encode(X_i), partition Y_i into N symbols: $Y_i[1:N]$;

  Broadcast$(Y_i[i])$;

  Wait Until Receive All $Y_j[j], j \in [1:N] \setminus i$;

  $Y := Y_1[1] * \cdots * Y_N[N]$;

  If Y is undecodable

    Execute *Algorithm 4.1* ;

  Else

    $X :=$ Decode(Y);

    $F_i := (X = X_i)$;

    Broadcast(F_i);

    If $Majority(F_1, \cdots, F_N) = TRUE(1)$

      Return(X);

    Else

      Execute *Algorithm 4.1* ;

□

Notice that to execute *Algorithm 4.1*, each module P_i does not need to send its whole result X_i. It only needs to send additional $N - (d+t) - 1$ *symbols* of its codeword Y_i. Since the code is designed to detect up to d *and* correct up to t *symbols*, it can correct up to $d+t$ *erasures*, thus the unsent $d+t$ symbols of Y_i can be regarded as *erasures* and recovered; hence the original X_i can be decoded from Y_i.

The flow chart of the algorithm given in Figure 4.1 shows the algorithm more clearly. and the following example shows how the algorithm works.

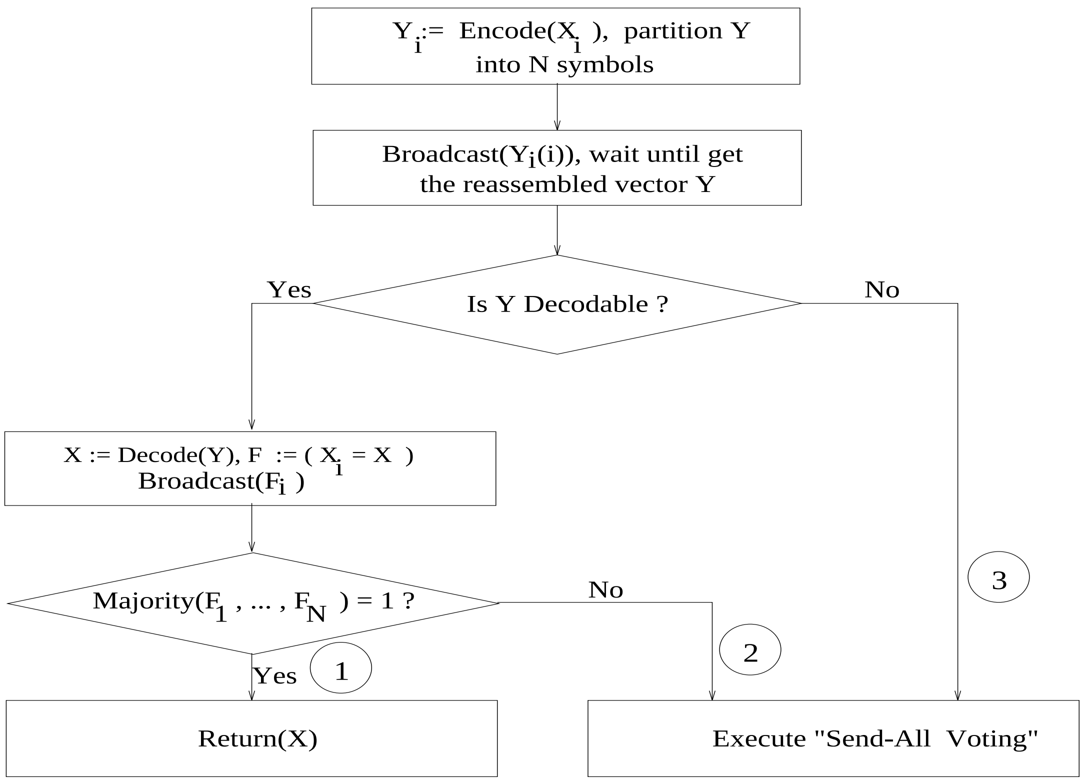

Figure 4.1: Flow chart of *Algorithm 4.3*

Example 4.3 *Voting Algorithm 4.3*

There are 5 modules in an NMR system, and the task result has 6 bits, i.e., $N = 5$ and $m = 6$. Here the EVENODD code[4] is used which divides 6-bit information into 3 *symbols* and encode information *symbols* into a 5-*symbol* codeword. This code can correct 1 error *symbol*, i.e., $d = t = 1$.

Now if $X_i = 000000$, $i = 1, 2, 3, 4$, and $X_5 = 000011$, then with the EVENODD code, $Y_i = 0000000000$, $i = 1, 2, 3, 4$, and $Y_5 = 0000111101$. After each module broadcasts 1 *symbol* (i.e., 2 bits) of its codeword, the reassembled vector is Y=0000000001. Since Y has only 1 error symbol, it can be decoded into X=000000. From the flow chart of the algorithm, we can see that $F_i = 1$, $i = 1, 2, 3, 4$, and $F_5 = 0$, thus $Majority(F_1, \cdots, F_5) = 1$, so X=000000 is the majority voting result.

In this case, there are 2 rounds of communication, and the communication complexity is 15 bits. As a comparison, *Algorithm 4.1* needs 1 round of communication, and its communication complexity is 30 bits; *Algorithm 4.2* needs 3 rounds of communication, and its communication complexity is 35 bits. In this example, the EVENODD code is used, but the specific code itself does not affect the communication complexity as long as it has same properties as the EVENODD code, namely, it is an *MDS* code with $d = t = 1$. □

As can be seen in the flow chart of the algorithm, introducing voting on the F_i's ensures that it is impossible to reach a *false* voting result, and going to the *Send-All Voting* in the worst case guarantees that the modules reach the majority result if it exists. Thus the algorithm gives a correct majority voting result. A rigorous correctness proof of the algorithm follows.

4.3.2 Correctness of the Algorithm

Theorem 4.1 Algorithm 4.3 *gives* $Majority(X_1, \cdots, X_N)$ *for a given set of local computational results* X_i*'s* $(i = 1, \cdots, N)$.

Proof: From the flow chart of the algorithm, it is easy to see that the algorithm terminates in the following two cases:

1. Executing the *Send-All Voting algorithm*: the correct majority voting result is certainly reached;

2. Returning an X: in this case, since $Majority(F_1, \cdots, F_N) = TRUE(1)$, i.e., majority of the $X's$ are equal to X, the correct majority voting result is also reached: X is the majority result. □

To see how the algorithm works for various instances of the local results X_i's $(i = 1, \cdots, N)$, we give two stronger observations about the algorithm, which also help to analyze the communication complexity of the algorithm.

Observation 4.1 *If* $Majority(X_1, \cdots, X_N) = \phi$*, then* Algorithm 4.3 *outputs* ϕ*, i.e.,* Algorithm 4.3 *never gives a* false *voting result.*

Proof: It is easy to see from the flow chart that after the first round of communication, each module gets the same *voting vector* Y. According to the decodability of Y, there are two cases:

1. If Y is undecodable, then the *Send-All Voting* algorithm is executed, and the output will be ϕ;

2. If Y is decodable, the decoded result X now can be used as a reference result. But since there does not exist a majority voting result, a majority of the X_i's are not equal to the X, i.e., $Majority(F_1, \cdots, F_N) = FALSE$, so the *Send-All Voting* algorithm is executed, and the output again will be ϕ. □

Observation 4.2 *If* $Majority(X_1, \cdots, X_N) = X(\neq \phi)$, *then* Algorithm 4.3*'s output is exactly X, i.e.,* Algorithm 3 *will not* miss *the majority voting result.*

Proof: Suppose there are e modules whose local results are different from the majority result X, then $e \leq \lfloor \frac{N-1}{2} \rfloor$.

1. If $e \leq t$, then there are e *error symbols* in the *voting vector* Y with respect to the corresponding codeword of the majority result X, so Y can be correctly decoded into X. A majority of the X_i's are equal to X, i.e., a majority of the F_i's are 1, hence the correct majority result X is outputted.

2. If $e > t$, then Y is either undecodable or incorrectly decoded into another X', where $X' \neq$ X. In either case, the *Send-All voting* algorithm is executed and the correct majority result X is reached. □

4.3.3 Proper Code Design

In order to reduce the communication complexity, we need an error-correcting code that can be used in practice for *Algorithm 4.3.* Consider a block code of length *M*. Because of the symmetry among the *N* modules, *M* needs to be a multiple of *N*, i.e., each codeword consists of *N symbols*, and each *symbol* has *k* bits, thus $M = Nk$. If the *minimum distance* of the code is d_{min}, then $d_{min} \geq (d+t)k+1$, where $0 \leq t \leq d \leq \lfloor \frac{N}{2} \rfloor$, since the code should be able to *detect* up to d error symbols and *correct* up to t error symbols[19]. Recall that the final voting result has m bits, the code to design is a $(Nk, m, (d+t)k+1)$ block code.

To get the smallest value for k, by the *Singleton Bound* in coding theory[19],

$$Nk - m + 1 \geq (d+t)k + 1 \tag{4.1}$$

we get

$$k \geq \frac{m}{N-(d+t)} \tag{4.2}$$

Equality holds for all *MDS Codes*[19].

So, given a specified (d,t), the smallest value for k is $\lceil \frac{m}{N-(d+t)} \rceil$. If $\frac{m}{N-(d+t)}$ is an integer, all *MDS Codes* can achieve this lower bound of k. One class of commonly used MDS codes for arbitrary distances is the Reed-Solomon code[19]. If $\frac{m}{N-(d+t)}$ is not an integer, then *any* $(Nk, m, (d+t)k+1)$ block code can be used, where $k = \lceil \frac{m}{N-(d+t)} \rceil$. One example of this is the BCH code, which can also have arbitrary distances[19]. The exact parameters (k,d,t) can be achieved by *shortening* (i.e., setting some information symbols to zeros) or *puncturing* (deleting some parity symbols) proper codes[19].

Notice that $0 \leq t \leq d \leq \frac{N-1}{2}$, thus $\frac{m}{N} \leq k \leq m$. In most applications, $N \ll m$, thus the N bits of F_i's can be neglected, and k is approximately the number of the bits that each module needs to send to get final voting result, so the communication complexity of *Algorithm 4.3* is always lower than that of *Algorithm 4.1*.

In this chapter, only the communication complexity of voting is considered, since in many systems, computations for encoding and decoding on individual nodes are much faster than reliable communications among these nodes; the latter requires rather complicated data management in different communication stacks and retransmission of packets between distributed nodes when packet loss occurs. However, in real applications, design of proper codes should also make the encoding and decoding of the codes as computationally efficient as possible. When the distances of codes are relatively small, as in most applications when the error probability p is relatively low, more computationally-efficient MDS codes should be used, such as the X-Code or the B-Code discussed in the previous chapters.

4.4 Communication Complexity Analysis

4.4.1 Main Results

From the flow chart of *Algorithm 4.3*, we can see if the algorithm terminates in *Branch 1*, i.e., the algorithm gets a majority result, then the communication complexity is $N(k+1)$; if it terminates in *Branch 2*, then the communication complexity is $N(m+1)$; finally if the algorithm terminates in *Branch 3*, the communication complexity is Nm. Thus the *worst case* communication complexity C_w is $N(m+1)$. When $m \gg 1$, $C_w \approx Nm$.

Let C_a denote the *average case* communication complexity of *Algorithm 4.3*, and define the *average reduction factor* α as the ratio of C_a over the communication complexity of the

Send-All Voting algorithm, i.e. $\alpha = \frac{C_a}{Nm}$. The following theorem gives the relation between α and the parameters of both an NMR system and the corresponding code:

Theorem 4.2 *For an NMR system with N modules, each of which executes an identical task whose result is m-bits long. Assume each module has computational error probability p, independent of other modules' activities. If* Algorithm 4.3 *uses an ECC which can detect up to d and correct up to t error symbols, and $m \gg N > 1$, then the following relation holds for the average reduction factor of* Algorithm 4.3*:*

$$\alpha < \frac{P_1}{N-(d+t)} + (1-P_1) + \frac{1}{m} \tag{4.3}$$

where

$$P_1 = \sum_{i=0}^{t} \begin{pmatrix} N \\ i \end{pmatrix} p^i (1-p)^{N-i} \tag{4.4}$$

Proof: To get the average case communication complexity C_a of *Algorithm 4.3*, we need to analyze the probability P_i of the algorithm terminating in the *Branch i*, $i = 1, 2, 3$. First assume that if a module has an erroneous result X_i, then it contributes an error symbol to the voting vector Y. From the proof of *Observation 2*, if the algorithm terminates in *Branch 1*, then at most t modules have computational errors; the probability of this event is exactly P_1. The event that the algorithm reaches *Branch 2* corresponds to the *decoder error* event of a code with *minimum distance* of d+t+1, thus[19]

$$P_2 = \sum_{i=d+t+1}^{N} A_i \sum_{k=0}^{\lfloor \frac{d+t}{2} \rfloor} P_{ik} \tag{4.5}$$

where $\{A_i\}$ is the *weight distribution* of the code being used, and P_{ik} is the probability that a received vector Y is exactly Hamming distance k away from a *weight-i* (binary) codeword of the code. More precisely,

$$P_{ik} = \sum_{j=0}^{k} \begin{pmatrix} i \\ k-j \end{pmatrix} \begin{pmatrix} N-i \\ j \end{pmatrix} p^{i-k+2j} (1-p)^{N-i+k-2j} \tag{4.6}$$

If the weight distribution of the code is unknown, P_2 can be approximately bounded by

$$P_2 \leq 1 - \sum_{i=0}^{\lfloor \frac{d+t}{2} \rfloor} \begin{pmatrix} N \\ i \end{pmatrix} p^i (1-p)^{N-i} \tag{4.7}$$

since the second term in the right side of the inequality above is just the probability of event that *correctable* errors occur. Finally P_3 is the probability of the *decoder failure* event,

$$P_3 = 1 - P_1 - P_2 \tag{4.8}$$

Now notice that a module with an erroneous result can also contribute a correct symbol to the voting vector, so the average case communication complexity is

$$C_a \leq P_1 N(k+1) + P_2 N(m+1) + P_3 Nm \tag{4.9}$$

and the average reduction factor is

$$\alpha \leq \frac{k}{m} P_1 + (1 - P_1) + \frac{P_1 + P_2}{m} \tag{4.10}$$

By noticing that $k = \lceil \frac{m}{N-(d+t)} \rceil$, and $P_1 + P_2 < 1$, we get the result of Eq. (4.3). $\Box$

Remarks on the theorem : From Eq. (4.3), we can see the relation between the average reduction factor α and each branch of *Algorithm 4.3.* The first term relates to the first branch, whose reduction factor is $\frac{k}{m}$, or $\frac{1}{N-(d+t)}$ when m is large enough relative to N that the round-off error can be neglected; P_1 is the probability of that branch. One would expect this term to be the dominant one for α, since with a properly designed code tuned to the system, the algorithm is supposed to terminate at *Branch 1* in most cases. The second term is simply the probability that the algorithm terminates at either *Branch 2* or *Branch 3*; in this case the reduction factor is 1 (i.e., there is no communication reduction since all the local results are transmitted), without considering the 1 bit for F_i's in *Branch 2.* The last term is due to the 1 bit for voting on F_i's. When the local result size is large enough, i.e., $m \gg 1$, this 1 bit can be neglected in our model. Thus in most applications, the assumption that $m \gg 1$ is quite reasonable, and the result of the theorem can be simplified to

$$\alpha \approx \frac{P_1}{N-(d+t)} + (1 - P_1) \tag{4.11}$$

From the above theorem and its proof, it can be seen that for a given *NMR* system (i.e., N and p), P_1 is only a function of t. Once t is chosen, it is easy to see from Eq. (4.3) or Eq. (4.11) that α decreases monotonically as d decreases. Recall that $0 \leq t \leq d$, thus for a fixed t, setting $d = t$ minimizes α when $m \gg 1$. Even though it is not straightforward to get a closed form solution for the value of t which minimizes α, it is almost trivial to get such an optimal t by numerical calculation.

Figure 4.2 shows relationship between α and (t, p, N). Figure 4.2(a) and Figure 4.2(b) show how α (using Eq. (4.11)) changes with t for some fixed (N, p) and $d = t$. It is easy to see that for small p and reasonable N, a small t (e.g., $t \leq 2$, for $N \leq 51$ with $p = 0.01$) can achieve minimal α. These results show that for a highly reliable *NMR* system (e.g., $p \leq 0.01$), adding a small amount of redundancy to the local results and applying error-correcting codes to them can drastically reduce the communication complexity of the majority voting. Since the majority result is m bits long, and each module gets an identical result after voting, the communication complexity of the voting problem is at least m bits, thus $\alpha \geq \frac{1}{N}$, i.e., $\frac{1}{N}$ is a lower bound of α. Figure 4.2c shows how close *Algorithm 4.3* comes to achieving the theoretical lower bound of α.

(a) α vs. t for different p with fixed N = 31 (b) α vs. t for different N with fixed p = 0.01 (c) Minimum α vs. N

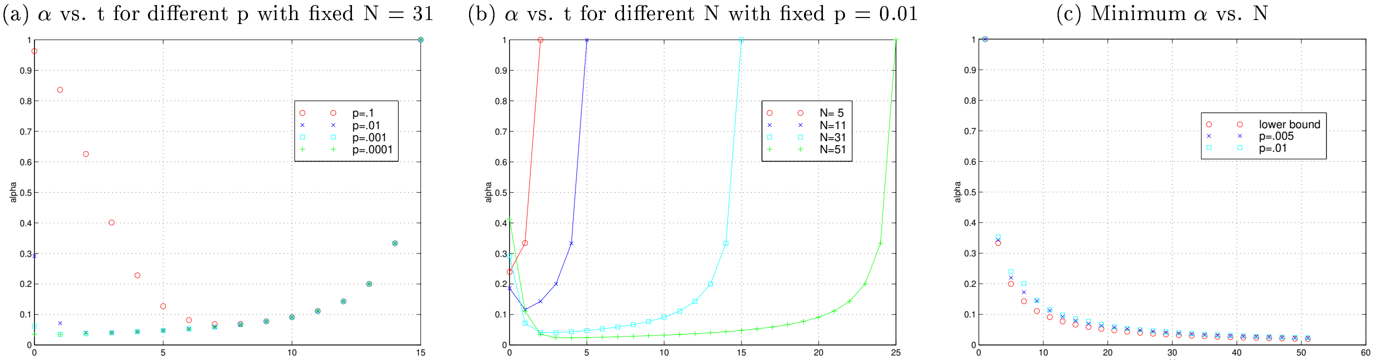

Figure 4.2: Relations between α and (t, p, N)

4.4.2 More Observations

From the above results, we can see that the communication complexity of *Algorithm 4.3* is determined by the code design parameters (d, t). In an *NMR* system with N modules, we only need to consider the case where at most $\lfloor \frac{N}{2} \rfloor$ modules have local results different from the majority result, thus the only constraint of (d,t) is $0 \leq t \leq d \leq \lfloor \frac{N}{2} \rfloor$. For some specific values of (d,t), the algorithm reduces to the following cases:

1. When $d = t = \frac{N-1}{2}$, i.e., a *repetition* code is used, the algorithm becomes *Algorithm 4.1*. Since a repetition code is always the worst code in terms of redundancy, it should always be avoided for reducing the communication complexity of voting. On the other hand, when d=t=0: the algorithm becomes *Algorithm 4.2*, and from Figure 4.2, we can see that for small enough p and reasonable N, e.g., $p = 10^{-5}$ and $N = 31$, *Algorithm 4.2* actually is the best solution of the majority voting problem in terms of the communication complexity. In addition, *Algorithm 4.2* has low computational complexity since it does not need any complex encoding or decoding operations. Thus the *ECC* voting algorithm is a generalized voting algorithm, and its communication complexity is determined by the code chosen.

2. $t = 0$, then the code only has detecting capability, but if $m \gg N$, then the analysis above shows increasing d actually makes α increase. Thus, when $m \gg N$, it is not good to add redundancy to the local results just for error detecting, i.e., using only *EDCs* (*error detecting codes*) does not help to reduce the communication complexity of voting. The scheme proposed in [21] ($d = \lfloor \frac{N}{2} \rfloor$) falls under this category.

3. $d = \lfloor \frac{N}{2} \rfloor$: as above, it is not good in general to have $d > t$ in terms of α, since increasing d will increase α for fixed t. But in this case, *Algorithm 4.3* has a special property: *Branch 2* of the algorithm can declare there is *no* majority result directly, *without* executing the *Send-All Voting* algorithm, because the code now can detect up to $\lfloor \frac{N}{2} \rfloor$ errors. So if there was a majority result, then Y (refer to Figure 4.1) can have at most $\lfloor \frac{N}{2} \rfloor$ erroneous modules, and since Y is decodable, the majority of the local results should agree with the decoded result X, i.e., $Majority(F_1, \cdots, F_N) = TRUE(1)$. This differs from the actual $Majority(F_1, \cdots, F_N)$, so there is *no* majority result. By setting d to $\lfloor \frac{N}{2} \rfloor$, *Algorithm 4.3* always has *2 rounds* of communication and the *worst case* communication complexity is thus Nm instead of $N(m + 1)$. This achieves the lower bound of the *worst case* communication

complexity of the distributed majority voting problem[21].

4.5 Experimental Results

In this section, we show some experimental results of the three voting algorithms discussed above. The experiments are performed on a cluster of Intel Pentium/Linux-2.0.7 nodes connected via a 100 Mbps Ethernet. Reliable communication is implemented by a simple improved UDP scheme: whenever there is a packet loss, the voting operation is considered a failure and is redone from the beginning. By choosing suitable packet size, there is virtually no packet loss using UDP.

To examine the real performance of the above three voting algorithms, N nodes vote on a result of length m using all three voting algorithms. For the *ECC Voting* algorithm, an *EVENODD Code* is used, which corrects 1 *error symbol*, i.e., $d = t = 1$. Random errors are added to local computing results with a preassigned error probability p, independent of results at other nodes in the NMR system. The performance is evaluated by two parameters for each algorithm: the total time to complete the voting operation T and the communication time for the voting operation C. The maximum T and C of the whole NMR system are chosen from among all the local T's and C's,, since if the voting operation is considered a collective operation, the system's performance is determined by the *worst* local performance in the system. For each set of NMR system parameters (N nodes and error probability p), each voting operation is done 200 times, and random computation errors in each run are independent of those in other runs. The arithmetic average of C's and T's are regarded as the performance parameters for the tested NMR system.

Experimental results are shown in Figures 4.3 through 4.5. Figure 4.3 compares the experimental *average reduction factors* of the voting algorithms with the theoretical results in the previous section, for an NMR system of 5 nodes. Figure 4.4 shows the performance (T and C) of the voting algorithms. Detailed communication patterns of the voting algorithms are shown in Figure 4.5 to provide some deeper insight into the voting algorithms.

Figure 4.3(a) and Figure 4.3(b) show the experimental *average reduction factors* of the voting communication time (C) for the *Simple Send-Part Voting* algorithm and the *ECC Voting* algorithm. Figure 4.3(a) and Figure 4.3(b) also show the theoretical average reduction factors of the Algorithm 4.2 and 4.3 as computed from Eq. (4.11). Notice that the average communication time reduction factors α of both *Algorithm 4.2* and *Algorithm 4.3* are below 1 for all the result sizes greater than 1 Kbyte, Also notice that as the computational result size increases, the reduction factor approaches the theoretical bound.

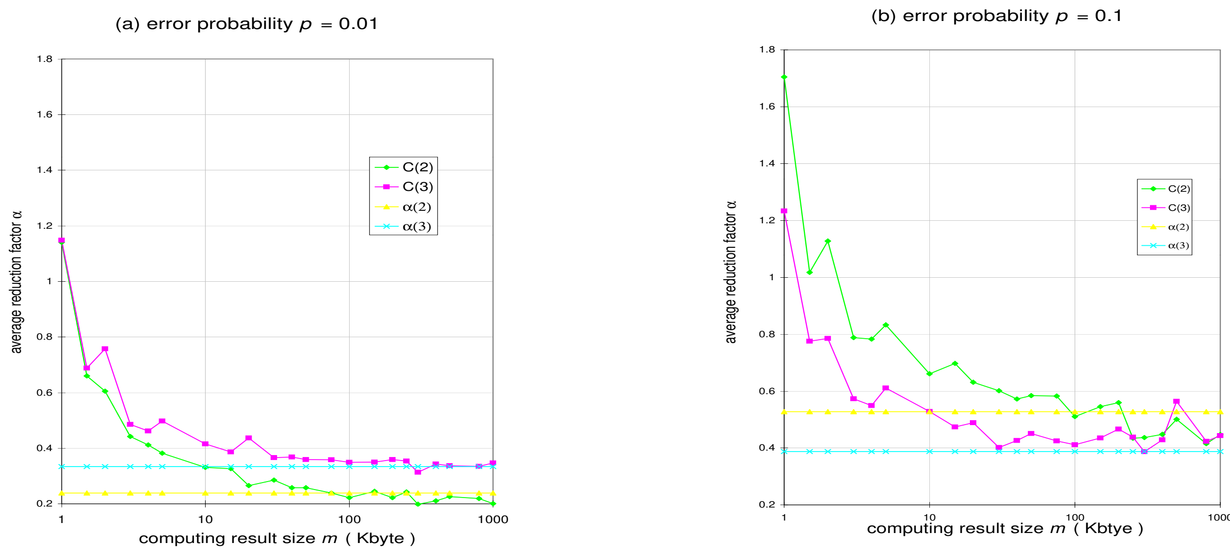

Figure 4.3: Average reduction factors

(C(i) is the experimental average reduction factor of communication time for voting using *algorithm 4.i*, and α(i) is the theoretical bound of the average communication reduction factor using *algorithm 4.i*, $i = 2, 3$)

Figure 4.4 shows the performance of each voting algorithm applied to an NMR system of 5 nodes. Figure 4.4(a)(b) show the *total* voting time T and Figure 4.4(c)(d) show the *communication* time C for voting. The only different parameter of the NMR systems related to Figures (a) and (b) (symmetrically (c) and (d)) is the error probability p: $p = 0.1$ in Figures (a) and (c), while $p = 0.01$ in Figures (b) and (d). It is easy to see from the figures that for *Algorithm 4.1 (Send-All Voting)*, T and C are the same, since there is no local computation other than communication. Figures 4.4(a)(b) show that Algorithms 4.2 (*Simple Send-Part Voting*) and 4.3 (*ECC Voting*) perform better than the Algorithm 4.1 (*Send-All Voting*) in terms of the total voting time T. On the other hand, Figures 4.4(c)(d) show that, in terms of the communication complexity C, the *ECC Voting* algorithm is better than the *Simple Send-Part Voting* algorithm when the error probability is relatively large (Figure 4.4(c)) and worse than the *Simple Send-Part Voting* algorithm when the error probability is relatively small (Figure 4.4(d)), which is consistent with the analysis results in the previous section.

In the analysis in the previous section, the size m of local computing result does not show up as a variable in the average reduction factor function α, since the communication complexity is considered to be only proportional to the size of the messages that need to be broadcast. But practically, communication time is *not* proportional to message size, since startup time of communication also needs to be included. More specifically, in the Ethernet environment, since the maximum packet size of each physical send (broadcast) operation is limited by the physical ethernet, communication completion time becomes a more complicated function of the message size. Thus from the experimental results, it can be seen that for a computing result of small size, e.g., 1 Kbyte, the *Send-All Voting* algorithm actually performs best in terms of both C and T, since the startup time dominates the performance of communication. Also, the communication time for broadcasting the 1-bit *voting flags* cannot be neglected as analyzed in the previous section, particularly for a computational result of small size. This can also be seen from the detailed voting communication time pattern in Figures 4.5(a)(b): *round 2* of the communication is for the 1-bit *voting flag*; even though it finishes in much more shorter time than *round 1*, it is still not negligibly small. This explains the fact that for small size computing results, the average communication time reduction factors of *Algorithm 4.2* and *Algorithm 4.3* are quite different from their theoretical bound.

Further examination of the detailed communication time pattern of voting provides a deeper insight into *Algorithm 4.3*. From Figures 4.5(c)(d), it is easy to see that in the first round of communication, *Algorithm 4.2* needs *less* time than *Algorithm 4.3*, since the size of the message to be broadcast is smaller for *Algorithm 4.2*. Also, the first round of communication time does not vary with the error probability p for the either algorithm. The real difference between the two algorithms lies in the third round of communication. From Figure 4.5(c), this time is small for the both algorithms since the error probability p is small (0.01). But as the error probability p increases to 0.1, as shown in Figure 4.5(d), for *Algorithm 4.2*, the third round time increases to more than the first round time, since it has no *error-correcting* capability. Once a full message needs to be broadcast, the message size is much bigger than that in the first round. On the other hand, though the communication time for the third round also increases for *Algorithm 4.3*, it is still much smaller than in the first round; this comes from the error-correcting codes that *Algorithm 4.3* uses, which can correct the most frequently occurring error pattern: errors at a single computing node. Thus, even though the computation error probability is high, in most cases, the most expensive third round of communication can be avoided. Thus *Algorithm 4.3* performs better (in terms of communication complexity or time) than *Algorithm 4.2* in high error probability systems, just as the analysis in the previous section predicted.

4.6 Summary

We have proposed a deterministic distributed voting algorithm using error-correcting codes to reduce the communication complexity of the voting problem in *NMR* systems. We have also given a detailed theoretical analysis of the algorithm. By choosing the design parameters (d, t) of the error-correcting code, the algorithm can achieve a low communication complexity, which is quite close to its theoretical lower bound. We have also implemented the voting algorithm over a network of workstations, and the experimental performance results match the theoretical analysis well. The algorithm proposed here needs 2 or 3 rounds of communication. It is left as an open problem whether there is an algorithm for the distributed majority voting problem with *average case* communication complexity less than Nm using *only* 1 round of communication.

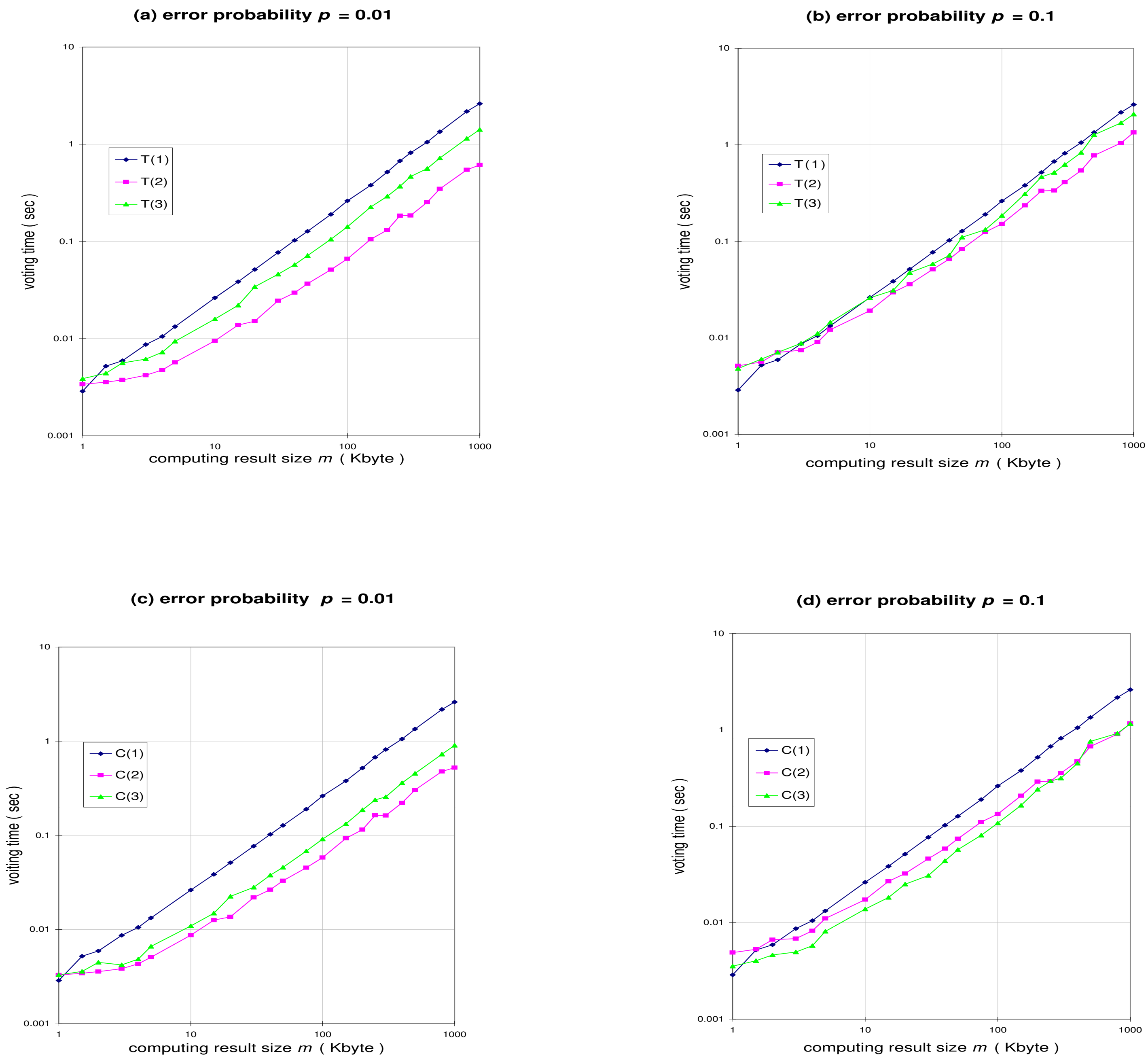

Figure 4.4: Experimental voting performance of 5-node NMR system

(T(i) and C(i) are the total and communication time for voting using *algorithm 4.i* respectively, $i = 1, 2, 3$)

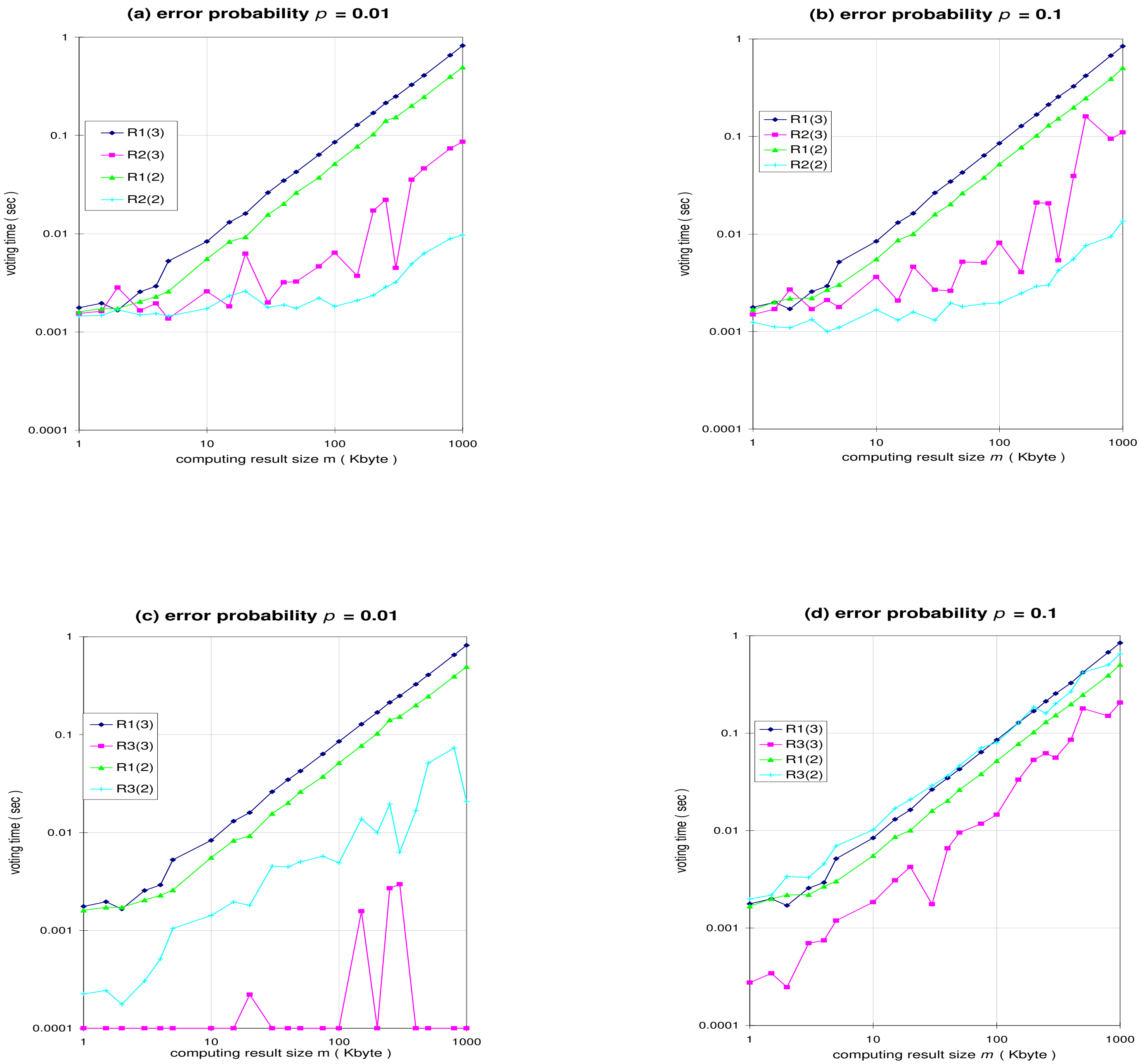

Figure 4.5: Detailed communication time pattern of voting

(R$i(k)$ is the communication time in *round* i using the voting *algorithm 4.k*, $i = 1, 2, 3$, and $k = 2, 3$)